

Programación 1



Tema 3

Sentencias de Control



Departamento de Ciencia de la
Computación e Inteligencia Artificial

Objetivos / Competencias

1. Conocer el concepto de algoritmo y entender la necesidad del diseño de algoritmos en el estudio y resolución de programas
2. Conocer y manejar con habilidad los tipos de sentencias de control existentes en un lenguaje de programación estructurado
3. Comprender la sintaxis y el funcionamiento de las diferentes sentencias de control en lenguaje C

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



El concepto de Algoritmo

Algoritmo

Secuencia ordenada de instrucciones que permiten resolver un problema en un número finito de pasos

Los algoritmos son independientes tanto del *lenguaje de programación* como del *ordenador* que los ejecuta

El concepto de Programa

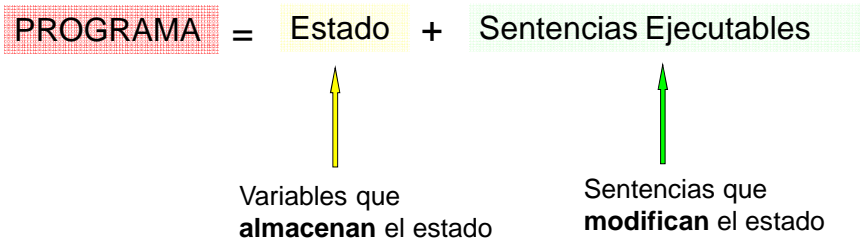
Programa

Conjunto de instrucciones (sentencias) ordenadas escritas en un lenguaje de programación para que una computadora lleve a cabo una determinada tarea

Los programas **codifican** algoritmos en un lenguaje de programación y son **ejecutados** en un ordenador

Estado de un programa

- El estado de un programa en un determinado instante es el valor que tienen cada una de sus variables en ese instante



Índice

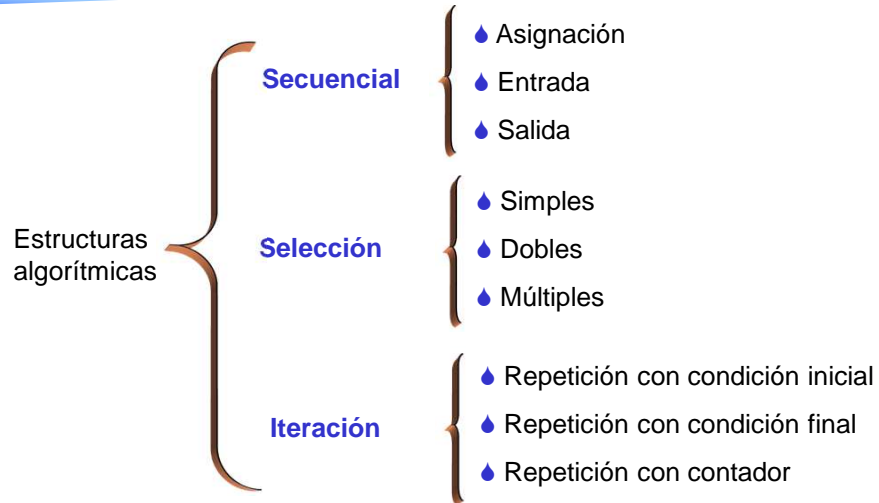
1. Algoritmos y Programas
2. **Estructuras algorítmicas**
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



Flujo de control de un algoritmo

- Define el orden que siguen las instrucciones del algoritmo
- Viene determinado por distintos tipos de **estructuras algorítmicas**

Tipos de estructuras algorítmicas

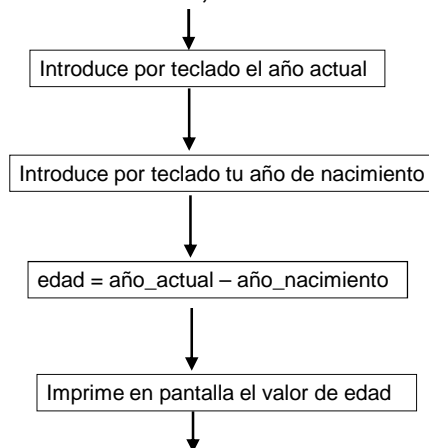


Programación 1. Dto. CCIA. Curso 2013-14

P-9

Estructura secuencial

- Las acciones (instrucciones) se efectúan una a continuación de otra, de manera consecutiva

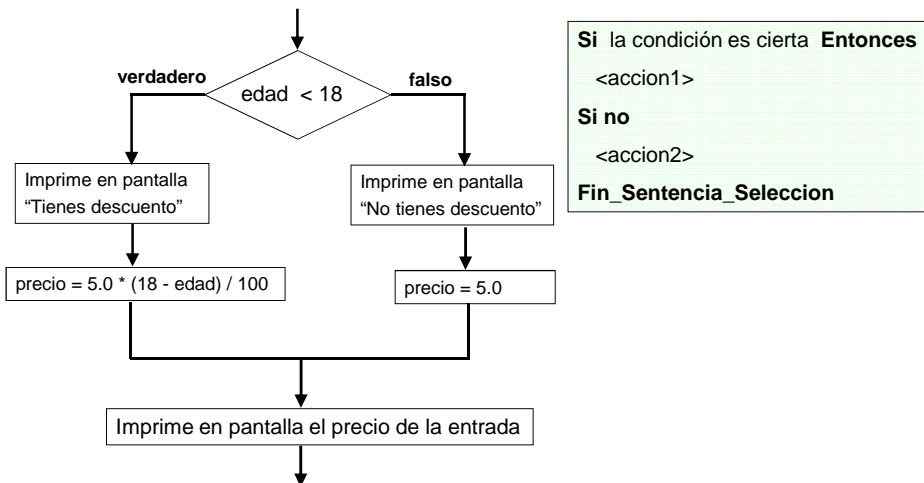


Programación 1. Dto. CCIA. Curso 2013-14

P-10

Estructura de selección

- Permite tomar decisiones entre distintas acciones alternativas dependiendo del valor de una condición

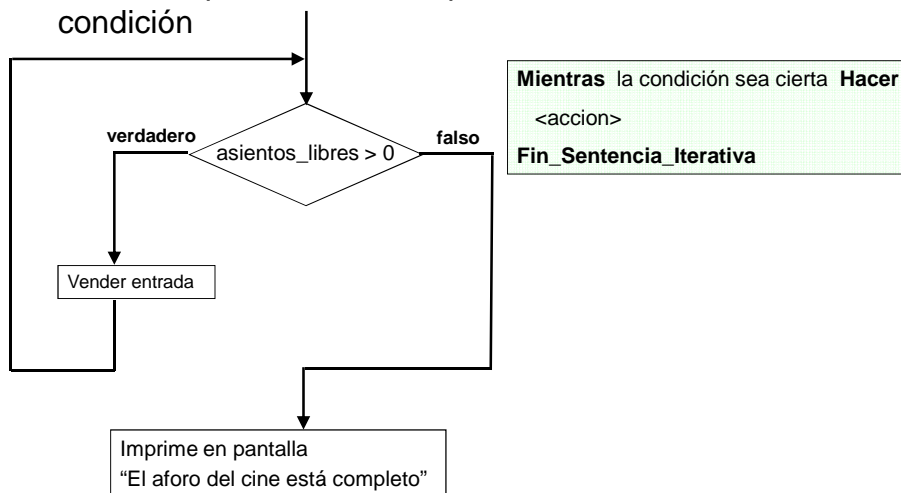


Programación 1. Dto. CCIA. Curso 2013-14

P-11

Estructura iterativa

- Permite repetir acciones dependiendo del valor de una condición



Programación 1. Dto. CCIA. Curso 2013-14

P-12

Índice

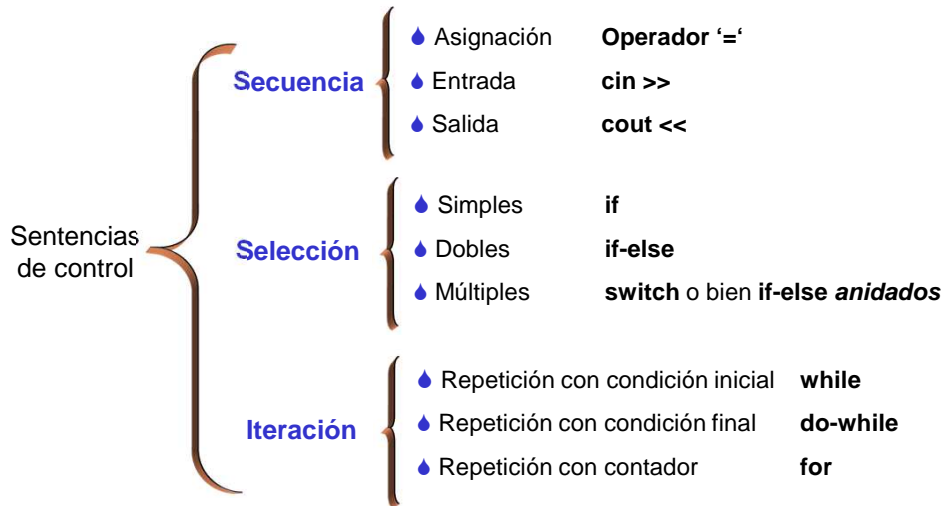
1. Algoritmos y Programas
2. Estructuras algorítmicas
- 3. Estructuras de programación**
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



Flujo de ejecución de un programa

- Define el orden en el que se ejecutan las diferentes sentencias que componen el programa
- Viene determinado por distintos tipos de **estructuras de programación** (sentencias de control)


Tipos de sentencias de control



Programación 1. Dto. CCIA. Curso 2013-14

P-15

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. **Sentencias secuenciales** 
5. Sentencias de selección
6. Sentencias de iteración
7. Traza de un programa
8. Estructura general de un programa
9. Fuentes de información

Programación 1. Dto. CCIA. Curso 2013-14

P-16

Sentencias secuenciales simples

◆ Sentencia de asignación

```
variable = valor ;
```

```
x = 20;  
y = 3;  
cociente = x / y;  
resto = x % y;
```

◆ Sentencia de entrada (LECTURA DE DATOS)

```
cin >> variable ;
```

```
cin >> x;  
cin >> y;
```

◆ Sentencia de salida

```
cout << dato ;
```

```
cout << "este texto se imprime en pantalla";  
cout << cociente;  
cout << "\n";
```

Secuencia de sentencias en lenguaje C y C++

- ◆ Una secuencia de sentencias puede estar constituida por N sentencias ($N \geq 0$)
- ◆ Cuando $N > 1$, la secuencia de sentencias **debe** estar encerrada entre **llaves**

```
{  
    secuencia de sentencias  
}
```

```
{ // inicio de secuencia de sentencias  
  
    cout << "Introduce dos números enteros";  
    cin >> x >> y;  
    cociente = x / y;  
    resto = x % y;  
    cout << "El cociente es: " << cociente << endl;  
    cout << "El resto es: " << resto;  
    cout << endl;  
  
} // fin de la secuencia de sentencias
```



Recuerda que todas las sentencias en C y C++ terminan con un punto y coma

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



Elegir una alternativa: Sentencia if

- ◆ Permite decidir si una secuencia de sentencias se van a ejecutar a continuación

```
if (expresión_lógica) {  
    secuencia de sentencias  
}
```

```
if (velocidad > 120) {  
    cout << "AVISO: te pueden multar";  
} // fin de la sentencia if
```

```
cout << "tu velocidad actual es: ";  
cout << velocidad << endl;
```

- ◆ Si el resultado de evaluar *expresión_lógica* es **verdadero** entonces se ejecuta la secuencia de sentencias asociada a la sentencia if
- ◆ Si el valor de *expresión_lógica* es **falso** entonces no se ejecuta la secuencia de sentencias y se pasará a ejecutar la sentencia siguiente al if



En C y C++, los **paréntesis** que encierran la expresión lógica del if son **obligatorios**

Elegir entre dos alternativas: Sentencia if-else

- ◆ Permite seleccionar entre dos secuencias de sentencias distintas

```
if (expresión_lógica) {  
    secuencia de sentencias 1  
}  
else {  
    secuencia de sentencias 2  
}
```

```
if (numero % 2 == 0) {  
    cout << "el número es par";  
}  
else {  
    cout << "el número es impar";  
} // fin de la sentencia if-else  
  
cout << endl;  
cout << "introduce otro número:";
```

- ◆ Si el valor de *expresión_lógica* es **verdadero** entonces se ejecuta la secuencia de sentencias situada a continuación del if (*secuencia de sentencias 1*)
- ◆ Si el valor de *expresión_lógica* es **falso** entonces se ejecuta la secuencia de sentencias situada a continuación del else (*secuencia de sentencias 2*)

Programación 1. Dto. CCIA. Curso 2013-14

P-21

Elegir entre múltiples alternativas: Sentencia if-else anidada

- ◆ Permite seleccionar entre múltiples secuencias de sentencias distintas

```
if (expresión_lógica_1) {  
    secuencia de sentencias 1  
}  
else if (expresión_lógica_2) {  
    secuencia de sentencias 2  
}  
else if (expresión_lógica_3) {  
    secuencia de sentencias 3  
}
```

```
if (nota >= 9 && nota <= 10)  
    cout << "tu nota es SOBRESALIENTE";  
else if (nota >= 7 && nota < 9)  
    cout << "tu nota es NOTABLE";  
else if (nota >= 5 && nota < 7)  
    cout << "tu nota es APROBADO";  
else if (nota >= 0 && nota < 5)  
    cout << "tu nota es SUSPENSO";  
else // última alternativa del if-else anidado  
    cout << "la nota es incorrecta";  
  
// aquí viene la siguiente sentencia después  
// del if-else anidado
```

- ◆ Sólo se ejecuta la primera de las secuencias de sentencias cuya expresión lógica asociada se evalúe a verdadero
- ◆ Si todas las expresiones lógicas se evalúan a falso
 - entonces se ejecuta la secuencia de sentencias situada a continuación del if-else anidado
 - a menos que la última alternativa este asociada a una parte else, en cuyo caso se ejecutará esta rama

Programación 1. Dto. CCIA. Curso 2013-14

P-22

Elegir entre múltiples alternativas: Sentencia switch

- ◆ Permite seleccionar entre múltiples secuencias de sentencias distintas

```
switch (expresión) {  
  case valor_1 : secuencia de sentencias 1;  
                break;  
  case valor_2 : secuencia de sentencias 2;  
                break;  
  case valor_3 : secuencia de sentencias 3;  
                break;  
  default      : secuencia de sentencias 4;  
}
```

```
switch (operador) {  
  case '+' : res = x + y;  
            break;  
  case '-' : res = x - y;  
            break;  
  case '*' : res = x * y;  
            break;  
  case '/' : res = x / y;  
            break;  
} // fin de la sentencia switch  
  
cout << "Resultado de la operación : ";  
cout << res;
```

- ◆ Sólo se ejecuta la secuencia de sentencias asociada al case cuyo valor se corresponda con el resultado de la expresión del switch
- ◆ Si el resultado de la expresión del switch no se corresponde con ningún valor de un case, se ejecuta la secuencia de sentencias asociada a la parte default (que es optativa)

Programación 1. Dto. CCIA. Curso 2013-14

P-23

Ejercicios

1. Escribe un programa que lea dos números distintos introducidos por teclado y muestre un mensaje de texto en pantalla indicando cuál es el mayor
2. Escribe un programa que solicite un número de segundos y muestre por pantalla dicha cantidad en horas, minutos y segundos
3. Escribe un programa que lea las coordenadas de tres puntos de un plano e indique si esos puntos forman un triángulo equilátero.
4. Escribe un programa que visualice tres opciones de un menú y permita al usuario seleccionar una de ellas, después de lo cual deberá aparecer un mensaje en la pantalla que muestre la opción seleccionada o bien un mensaje de error si la opción es incorrecta:

Ejemplo 1 de ejecución

```
1. Opción1 del menú  
2. Opción2 del menú  
3. Opción3 del menú
```

Introduce una opción del menú: 2
La opción seleccionada es 2

Ejemplo 2 de ejecución

```
1. Opción1 del menú  
2. Opción2 del menú  
3. Opción3 del menú
```

Introduce una opción del menú: 4
La opción seleccionada es incorrecta

Programación 1. Dto. CCIA. Curso 2013-14

P-24

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. **Sentencias de iteración**
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



Bucles

- Un **bucle** es una estructura de programación formada por una secuencia de sentencias, denominada **cuerpo** del bucle, que se puede repetir varias veces
- Cada ejecución del cuerpo del bucle es una **iteración**
- El número de veces que se ejecuta el cuerpo del bucle está controlado por una **condición** (expresión lógica)
- Por lo tanto, a la hora de diseñar e implementar un bucle, hay que tener en cuenta dos aspectos:
 1. **¿Cuál debe ser el cuerpo del bucle?**
 2. **¿Cuántas veces debe iterarse el cuerpo del bucle?**

Tipos de bucles

- En función de dónde se encuentra la condición que controla la ejecución del cuerpo del bucle, se distinguen los siguientes tipos de bucles:

- ❑ Bucles con condición inicial
 - Sentencia while
 - Sentencia for (repetición con contador)
- ❑ Bucles con condición final
 - Sentencia do-while

Repetición con condición inicial: Sentencia while

- Permite repetir cero o más veces la ejecución de una secuencia de sentencias mientras la condición sea verdadera

```
while (expresión_lógica) {  
    secuencia de sentencias  
}
```

```
caramelos=0;  
cout << "¿Quieres un caramelo?:";  
cin >> res;  
while (res == 'S' || res == 's') {  
    caramelos=caramelos +1;  
    cout << "¿Quieres otro caramelo?:";  
    cin >> res;  
} // fin de la sentencia while  
cout << "Te he dado " << caramelos << "caramelos";
```

- Mientras el resultado de evaluar *expresión_lógica* sea **verdadero** se ejecutará repetidamente la secuencia de sentencias (cuerpo del bucle)

Repetición con condición final: Sentencia do-while

- ◆ Permite repetir una o más veces la ejecución de una secuencia de sentencias mientras la condición sea verdadera

```
do {  
    secuencia de sentencias  
} while (expresión_lógica);
```

```
do {  
    cout << "Introduce la opción del menú:";  
    cin >> opcion;  
} while (opcion < 1 || opcion > 4);
```

- ◆ Primero se ejecuta el cuerpo del bucle (secuencia de sentencias) y después se evalúa la expresión lógica
- ◆ Mientras el resultado de evaluar *expresión_lógica* sea verdadero se ejecutará repetidamente el cuerpo del bucle

Repetición con condición inicial: Sentencia for

- ◆ Permite repetir un número determinado de veces la ejecución de una secuencia de sentencias
- ◆ El número de iteraciones del bucle es controlado por una variable usada como un **contador**

```
for (inicialización contador; expresión_lógica ; incremento contador) {  
    secuencia de sentencias  
}
```

```
for (i = 1; i <= 10; i++) {  
    cout << "Esta es la iteración " << i;  
}
```



En C y C++, la sentencia **i++** es una sentencia de asignación que equivale a **i = i+1**

El incremento del contador no tiene por qué ser de uno en uno, por ejemplo podría ser: **i = i+2;** **i = i*2;** ...

También se puede decrementar el contador: **i = i-1**

¿Cómo funciona la sentencia for?

Paso 1. Se ejecuta la sentencia de inicialización del contador
(**una sola vez**)

Paso 2. Se evalúa la expresión lógica de forma que :

- ☐ Si su valor es **verdadero** Entonces se ejecuta el cuerpo del bucle
- ☐ Si su valor es **falso** Entonces la sentencia for **FINALIZA** su ejecución

Paso 3. Después de ejecutarse el cuerpo del bucle, se ejecuta la sentencia de incremento del contador

Paso 4. Volver al paso 2.

El bucle for y su equivalente bucle while

💧 Cualquier bucle for se puede reescribir como un bucle while

```
for (expresión_1 ; expresión_2 ; expresión_3) {  
    secuencia de sentencias;  
}
```

```
for ( i = 1; i <= 10; i++) {  
    cout << "Esta es la iteración " << i;  
}
```



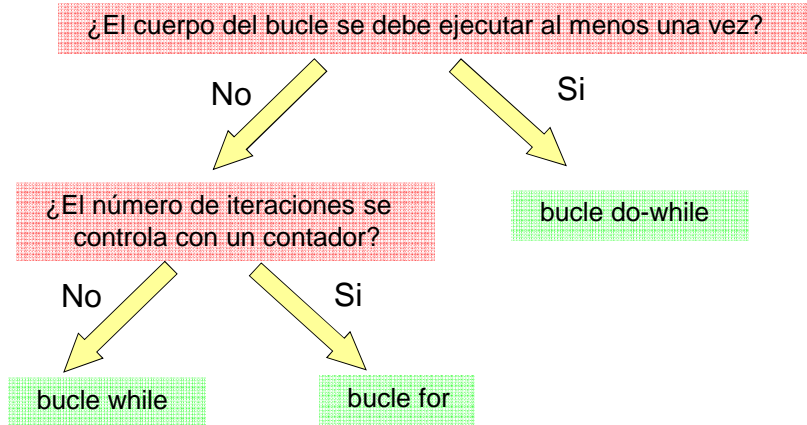
Se puede reescribir como



```
expresión_1 ;  
while (expresión_2) {  
    secuencia de sentencias;  
    expresión_3;  
}
```

```
i = 1;  
while ( i <= 10) {  
    cout << "Esta es la iteración " << i;  
    i++;  
}
```


¿Qué tipo de bucle tengo que usar?



Programación 1. Dto. CCIA. Curso 2013-14

P-33

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
- 7. Comentarios**
8. Traza de un programa
9. Estructura general de un programa
10. Fuentes de información



Programación 1. Dto. CCIA. Curso 2013-14

P-34

Comentarios en el código fuente

- ◆ Son notas aclaratorias que podemos incluir en un programa
- ◆ Facilitan el mantenimiento del programa
- ◆ En lenguaje C++ se utilizan los símbolos //

- ☐ Todo el texto entre // y el final de línea es un comentario

```
float nota_parcial; // nota de un parcial (dato de entrada)
// Calcular la nota media e imprimirla por pantalla
```

- ◆ En lenguaje C y C++ se también se utilizan los símbolos /* */

- ☐ Todo el texto encerrado entre /* */ se considera un comentario

```
/* Introducir las notas de todos los parciales y sumarias
(sólo cuando el dato introducido sea correcto) */
```

¿Cómo debe de ser un comentario?

- ◆ Un comentario debe explicar de forma clara y concisa **qué** es lo que hace una sección del código de un programa, y **no cómo** lo hace (para eso ya está el propio código)

Dado el siguiente código :

```
for (x=1; x <= 10; x++)
for (y=1; y<=10; y++)
cout << x << "*" << y << " = " << x*y << endl;
```

¿Qué opinas del siguiente comentario para dicho código?

```
/* Tenemos 2 bucles for anidados que se repiten 10 veces cada uno
En el bucle interno se imprime mensaje de texto en la pantalla que
indica el producto de las dos variables usados como contador en los bucles for
En total se imprimen 100 líneas en la pantalla */
```



¿qué comentario sería adecuado?

¿Dónde debo incluir un comentario y cuántos incluyo?

¿Dónde debo incluir un comentario?

- ◆ en la definición de un módulo (qué hace el módulo)
- ◆ al principio de una sección de código que realice una acción importante y no sea obvio lo que hace
- ◆ al principio del programa (una cabecera con el nombre del programa, autor, fecha, descripción del programa, etc.)

¿Cuántos comentarios debo incluir en mi programa?

- ◆ Demasiados comentarios son tan malos como muy pocos

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
- 8. Traza de un programa**
9. Estructura general de un programa
10. Fuentes de información



Concepto de traza de un programa

- Es la **secuencia de estados** por los que pasa un programa, es decir, el valor que van tomando las variables a medida que se va ejecutando el programa
- La traza se lleva a cabo mediante la **ejecución manual** de forma secuencial de las sentencias que componen el programa
- Las trazas se utilizan principalmente para **depurar** un programa



Las variables almacenan el estado de un programa y las sentencias ejecutables modifican dicho estado

Depurar un programa consiste en corregir errores de ejecución detectados al probarlo

Programación 1. Dto. CCIA. Curso 2013-14

P-39

Depurar un programa con una traza

```
// Dado un número N > 0, calcula la suma de todos
// los números impares menores que N

#include <iostream>
using namespace std;

main() {
    int num; // número leído (dato de entrada)
    int suma; // resultado del sumatorio (dato de salida)
    int i; // contador de bucle (dato auxiliar)

    cout << "Introduce un número mayor que cero:";
    cin >> num;

    // calcular el sumatorio e imprimirlo por pantalla
    suma = 0;
    for (i=1; i < num; i++) {
        if ( (num % 2) != 0 )
            suma = suma + i;
    }
    cout << "Es resultado es: " << suma << endl;
}
```

	num	suma	i
cin	5		
Inicializa suma	5	0	
Inicializa contador	5	0	1
1ª iteración del for	5	1	1
Incremento contador	5	1	2
2ª iteración del for	5	3	2
Incremento contador	5	3	3
3ª iteración del for	5	6	3
Incremento contador	5	6	4
4ª iteración del for	5	10	4
Incremento contador	5	10	5



¿Por qué no funciona?

Programación 1. Dto. CCIA. Curso 2013-14

P-40

Conocer qué hace un programa usando una traza

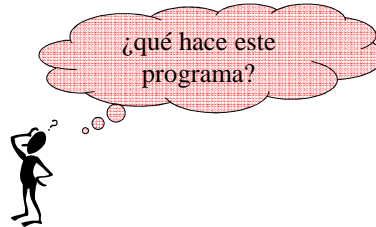
💧 También puedes utilizar una traza para comprender qué hace un programa o parte del código del mismo

```
#include <iostream>
using namespace std;

main() {
    float a, r;
    int b, i;

    cout << "Introduce un número real:";
    cin >> a;
    cout << "Introduce un número entero:";
    cin >> b;
    r = 1;
    for (i=0; i < b; i++)
        r = r * a;

    cout << "El resultado es: " << r << endl;
}
```



Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Trazas de un programa
- 9. Estructura general de un programa**
10. Fuentes de información



¿Qué tipo de programas debo ser capaz de hacer?

#directivas del preprocesador

Declaración de constantes

main() {

Declaración de variables:

de tipos simples

Cuerpo principal (sentencias ejecutables)

sentencias de Entrada y Salida

sentencias de asignación

sentencias de selección

sentencias iterativas

}

Programación 1. Dto. CCIA. Curso 2013-14

P-43

Ejemplo de programa

```
#include <iostream>
using namespace std;

const int NUM_PARCIALES = 5; // Número de exámenes parciales

main() {
    float nota_parcial; // nota de un parcial (dato de entrada)
    float suma; // suma total de notas (dato auxiliar)
    int i; // contador del bucle for (dato auxiliar)
    bool nota_incorrecta; // true si la nota introducida es incorrecta (dato auxiliar)
    float nota_final; // nota media de todos los parciales (dato de salida)

    suma = 0;
    // Introducir las notas de todos los parciales y sumarmas (sólo cuando el dato introducido sea correcto)
    for (i=1; i <= NUM_PARCIALES; i++) {
        do {
            cout << "Dime tu nota del parcial " << i << " ";
            cin >> nota_parcial;
            nota_incorrecta = (nota_parcial < 0.0 || nota_parcial > 10.0);
            if (nota_incorrecta)
                cout << "La nota introducida es incorrecta" << endl;
        } while (nota_incorrecta);
        suma = suma + nota_parcial;
    }
    // Calcular la nota media e imprimirla por pantalla
    nota_final = suma / NUM_PARCIALES;
    cout << "Tu nota final es: " << nota_final << endl;
}
```

Programación 1. Dto. CCIA. Curso 2013-14

P-44

Normas para un buen estilo de escritura de programas

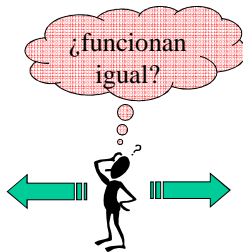
- ◆ Los nombres de variables y constantes deben sugerir su uso
- ◆ Utilizar saltos de línea entre partes que por su lógica deben considerarse por separado
- ◆ Sangrar el código

```
#include <iostream>
using namespace std;

main() {
    float base, potencia;
    int exponente, i;

    cout << "Introduce un número real:";
    cin >> base;
    cout << "Introduce un número entero:";
    cin >> exponente;
    potencia = 1;
    for (i=0; i < exponente; i++)
        potencia = potencia * base;

    cout << "Potencia: " << potencia << endl;
}
```



```
#include <iostream>
using namespace std;

main() { float a, r; int
    b, i; cout <<
    "Introduce un número real:"; cin >>
    a; cout <<
    "Introduce un número entero:";
    cin >> b; r
    = 1; for
    (i=0
    ; i < b;
    i++)    r = r * a;
    cout << "Es resultado es: " << r <<
    endl; }
```



Un programa escrito con un buen estilo de programación es más fácil de leer (más legible) y más fácil de modificar (más mantenible)

Programación 1. Dto. CCIA. Curso 2013-14

P-45

Ejercicios

5. Después de ejecutar cada uno de los siguientes fragmentos de programa, ¿cuál será el valor final de la variable x en cada uno de los casos?

Caso A

```
x = 0;
n = 16;
while ( n != 0 ) {
    x = x + n;
    n = n / 2;
}
```

Caso B

```
z = 12;
x = 0;
if ((z % 4) == 0)
    for (j = 0; j < 10; j + 4)
        x = x + j;
else
    for (j = 0; j < 10; j + 2)
        x = x + j;
```

6. Escribe un programa que lea números positivos y nos muestre el valor de su suma y la cantidad de números leídos.
7. Escribe un programa que lea un número entero mayor que cero y muestre por pantalla todos los divisores de dicho número
8. Modifica el programa del ejercicio 4 para añadir una cuarta opción que sea SALIR. El programa debe mostrar el menú continuamente, después de que el usuario elija opción, hasta que se elija la opción 4.

Programación 1. Dto. CCIA. Curso 2013-14

P-46

Índice

1. Algoritmos y Programas
2. Estructuras algorítmicas
3. Estructuras de programación
4. Sentencias secuenciales
5. Sentencias de selección
6. Sentencias de iteración
7. Comentarios
8. Traza de un programa
9. Estructura general de un programa

10. Fuentes de información



Programación 1. Dto. CCIA. Curso 2013-14

P-47

Bibliografía Recomendada


Fundamentos de Programación
Jesús Carretero, Félix García, y otros
Thomson-Paraninfo 2007. ISBN: 978-84-9732-550-9

 Capítulo 5

Problemas Resueltos de Programación en Lenguaje C
Félix García, Alejandro Calderón, y otros
Thomson (2002) ISBN: 84-9732-102-2

 Capítulo 3

Resolución de Problemas con C++
Walter Savitch
Pearson Addison Wesley 2007. ISBN: 978-970-26-0806-6

 Capítulo 2 (Apartado 2.4)

 Capítulo 7

Programación 1. Dto. CCIA. Curso 2013-14

P-48